A SMART MEMORY ARRAY PROCESSOR FOR TWO LAYER PATH FINDING*

Christopher R. Carroll, Caltech

This paper describes three examples of hardware implementations of path finding schemes based on the Lee-Moore maze solving algorithm. One is purely a demonstration circuit to show the technique. The other two are complete LSI implementations which should be usable in building large and useful path finding machines. One of these two LSI circuits, known as the *MAZER*, is designed to find shortest paths from one point to another on a plane, where there is only ***one*** layer of allowable routes to take. As its name suggests, this chip solves ordinary mazes, or on a more practical level, it can route wires on a one sided printed circuit board. The other LSI circuit, known as the *PATHFINDER*, is designed to handle the two sided printed circuit board case. It finds a ***least costly*** path from one point to another where there are ***two*** parallel planes on which routes are allowed. Crossing of the path from one plane to another can be either unrestricted, as in ***free via*** printed circuit boards, or permitted only in certain places, as in ***fixed via*** boards. The phrase "least costly" above can, for now, be read as "shortest", although in a later section a more general definition will be revealed.

The remainder of this document is divided into three parts. The first section outlines the original ***Lee-Moore algorithm*** for path finding, on which the circuits described later are based. The second section details the ***one layer hardware***, including both the ***demonstration circuit*** and the *MAZER* chip. Finally, the third section describes the *PATHFINDER* chip and the techniques used to conquer the problems encountered in two layer path finding. Documentation on the integrated circuits includes those results of testing and characterization which were available at the time of this writing.

---

*The research described in this paper was sponsored by the Defense Advanced Research Projects Agency, ARPA Order number 3771, and monitored by the Office of Naval Research under contract number N00014-79-C-0597.

1. THE LEE-MOORE ALGORITHM

The ***Lee-Moore algorithm*** for path finding, proposed by E. Moore in 1959 (1) and extended by C. Lee in 1961 (2), is a scheme for finding the shortest route between two points in a plane, where the route is composed of some number of vertical and horizontal segments through a rectangular grid superimposed on the plane.* This has been a popular algorithm for people doing problems related to maze solving because it is easy to implement and because it guarantees that a path will be found if one exists. The drawbacks to the algorithm are that it is expensive computationally in both time and space. However, the use of the hardware described in this paper circumvents these difficulties.

Suppose that the size of the grid, i.e. the pitch of the cells defined by the grid, is set to the minimum path width that is allowed. In the case of printed circuit board design this would be the minimum center to center spacing for adjacent wires. Suppose also that the grid is uniform and symmetric, forming an array of square cells, each a path width on a side. The path found by the algorithm from point A to point B will consist of a route beginning at the cell containing point A, continuing to a neighbor of that cell, and then to a neighbor of that second cell, and so on from a cell to one of its neighbors, until eventually the path ends in the cell containing point B. Some of the cells in the array may be ***blocked***, preventing the path from running through these cells. These would be "barriers", or "walls" in a maze, or cells occupied by previously routed wires in the printed circuit board application.

The algorithm finds the shortest path from A to B in two phases. One word of storage, which I will call the "label", is associated with each of the cells in the array. The first phase, called the ***Propagation Phase***, stores information in the labels throughout the array. The second phase, called the ***Retrace Phase***, then uses that information to find the required path.

---

* Since our geometry here is based on this grid, the distances mentioned will be ***Manhattan distances***, i.e. the distance from A to B would be the shortest distance covered by a taxi driver driving from A to B on the streets of Manhattan.

The Propagation Phase, which distributes the information, executes the following program:

```
put label -1 in all cells which are blocked
put label 0 in all cells which are not blocked
N:=1
put label N in the cell containing point A
while cell containing point B is labelled 0 and more activity is possible do
begin
  for every neighbor of every cell labelled N do
    if that neighbor is labelled 0 then label it (N+1) else leave it alone.
  N:=N+1
end
```

This part of the algorithm is illustrated in Figure 1. The purpose of this phase is to distribute information to the cells which can then be used to find the direction back to point A. The information is spread out in a propagating wavefront centered on point A, much like waves propagating away from a stone dropped in a pond. It is interesting to note that the only activity that takes place occurs at the frontier of this expanding wavefront. Cells ahead of the frontier merely wait for the wave to arrive, keeping their label of 0. Cells behind the frontier have already received the information they need, and simply keep it stored in their label.

The ***Retrace Phase***, using the information stored in the labels, executes the following program to find the path:

```
Start the path at the cell containing point B
N:= label of cell containing point B
if N=0 then there is no path from point A to point B
else begin
  While path has not yet reached cell containing point A do
  begin
    N:=N-1
    Continue the path to a neighbor of the current cell which contains
      the label N
  end
end
```

This part of the algorithm is illustrated in Figure 2. Notice that there is nothing to specify which cell to choose when there are two or more possible choices. This merely means that there are multiple paths between A and B that have the same length. To first order, there is thus no preference of one path over another, so no selection mechanism need be used. In practice, some scheme is often employed when there is a choice of paths to take. A common selection scheme is to avoid changing

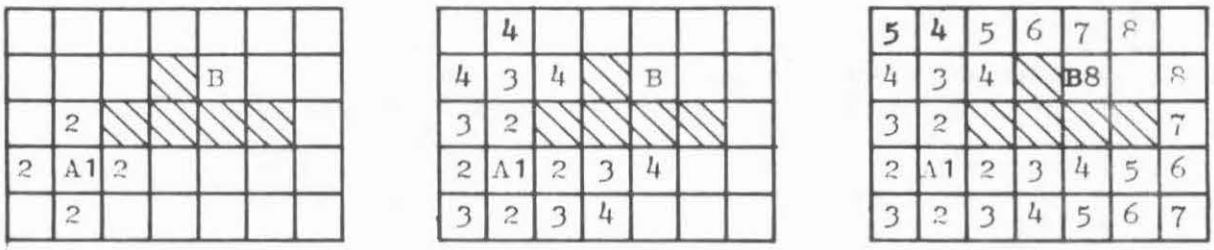

Figure 1. The Lee-Moore Propagation Phase

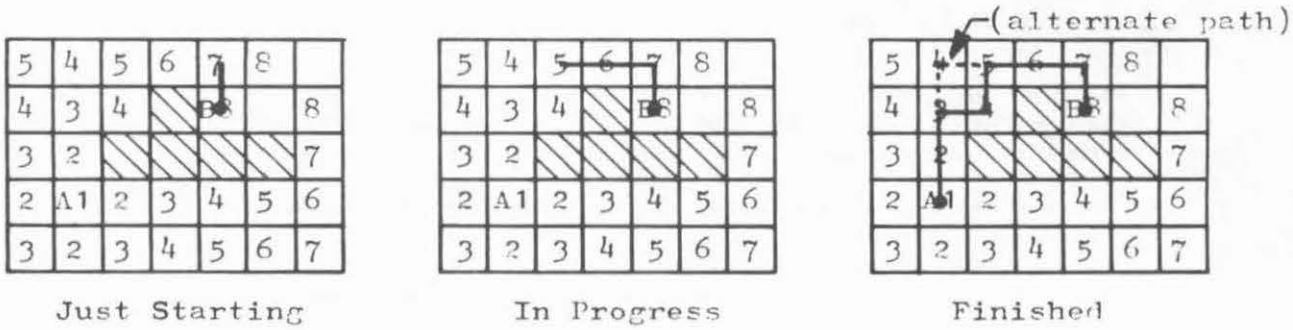

Figure 2. The Lee-Moore Retrace Phase

directions in the path during the Retrace Phase when it is unnecessary. This tends to minimize the number of bends in the resulting path. When this phase is complete, the algorithm either has found the required path from A to B or has proven that no such path exists.

Before beginning the discussion of the hardware implementations of this algorithm, a couple of things should be noted. First, examine the time complexity of the programs above. The Retrace Phase merely traces the path from B back to A using information stored in the cells. The only cells accessed are those along the selected path and their immediate neighbors. The time complexity is thus ***linear*** with respect to the path length. However, in the Propagation Phase, the situation is worse. Information is propagated in all directions around point A. The number of cells accessed is approximately proportional to the square of the path length. Thus, the time complexity here is ***quadratic*** with respect to the path length, making the algorithm as a whole quadratic. This is unfortunate, since for maze solving to be interesting, a large maze must be involved. In the circuit board application, for example, a cell array containing 1000 x 1000 cells would be common. The quadratic time aspect of the algorithm thus is a real handicap. Current software using this algorithm to route typical printed circuit boards can consume several hours of CPU time on a full-size computer. On top of that, the space requirement is also large. Circuit board routing requires 10-12 bits of storage for each cell, and a million 12 bit words is a lot of memory. So, both the space and time complexity of the algorithm need to be attacked in any successful hardware implementation. The next section shows how this was accomplished.

2. HARDWARE FOR FINDING ONE-LAYER PATHS

Implementing the Lee-Moore algorithm in hardware is a clean and natural thing to do. Because the problem is cellular and because information flows only between adjacent cells without using any long distance communication paths, the task is a natural one for an array processor structure with one processor per cell in the array. However, if there is to be any hope of building a large machine this way, there are two problems which must be overcome. First, ***the amount of storage per cell*** must be limited. In the original algorithm, in an array of unbounded size each cell would be required to contain an unbounded number of bits. Second, the ***global state*** required in the original algorithm, which was represented by "N" in the programs above, must be eliminated. Accomplishing these goals would result in a machine which could be extended to any size needed without undue complications.

The first goal, that of limiting the amount of storage required in each cell, was attacked by S. Akers in 1967 (3). He showed that only two bits were required per cell to implement the algorithm proposed by Lee and Moore. Of the four states available from the two bits, one indicated that a cell was blocked and unavailable for new paths. Another was used to indicate a cell that was so far untouched by the propagation process, like label 0 in the above programs. Then, instead of using the ascending ordinal numbers to label successive wavefronts in the propagation, Akers used successive members from the sequence

1, 1, 2, 2, 1, 1, 2, 2, 1, 1, 2, 2, . . .

These last two states stored the information necessary to get back to the point where the propagation started. See Figure 3. It was only required that the program remember whether it was on the first 1, second 1, first 2, or second 2 in the sequence when it stored a number in the goal cell containing point B. For example, if the program knew it was on the second 1 of a pair when it reached the goal, then in the Retrace Phase it looked for cells containing the above sequence in reverse order, starting with the first 1, then 2,2,1,1, etc., until it reached the starting cell. This was a big step forward, for only two bits of storage were needed for each cell, no matter how big the array of cells was made. Unfortunately this did nothing to solve the problem of having to distribute the next member of the sequence as a global variable to all the cells in the array.

The solution to the dilemma of global state is to use a slightly different strategy in the algorithm. This idea was recorded in an internal document of the Caltech Computer Science Department by Ivan Sutherland (4). Rather than numbering successive wavefronts with some sequence and then searching for the reverse of that sequence to find the path, simply store in each cell arrows which point to the neighbor(s) from which the wavefront approaches as it passes over that cell, and then just let the arrows show the path back to the starting cell. This approach is shown in Figure 4. Since the wavefront may reach a cell from more than one neighbor simultaneously, and since that fact is important when trying to select one of several equally good paths, an arrow for each neighbor is needed. These arrows require one bit each, because the wavefront either came from that neighbor or it didn't. Additionally, one bit is required to identify a cell as being blocked. Five bits per cell is more than Akers' two, but it is still a small number, and more importantly, it is still a

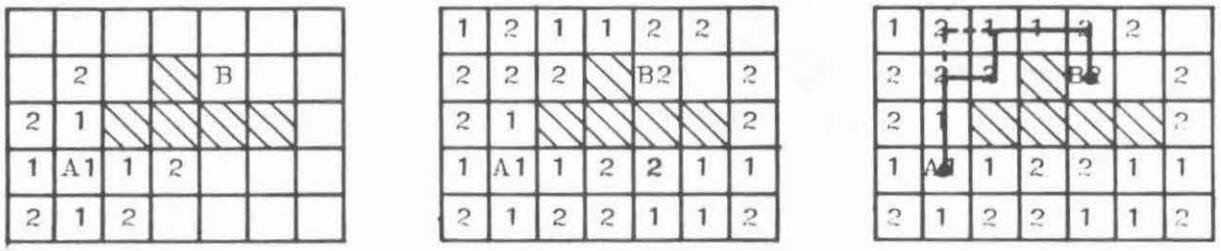

Figure 3. Akers' Modification

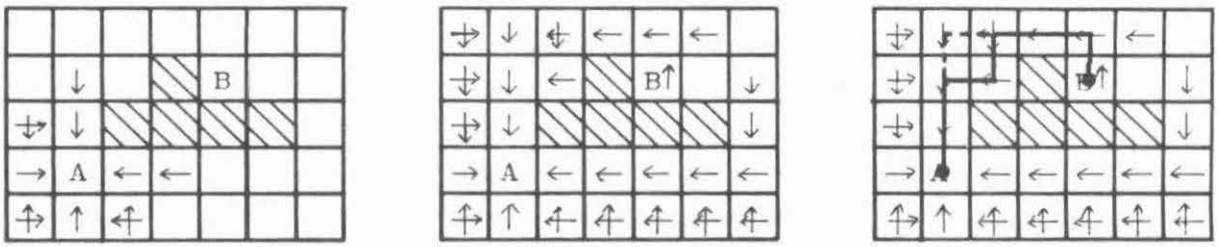

Figure 4. Modification for MAZER

bounded number, that does not change as the array of cells grows. With these modifications to the algorithm, the move to hardware is hardly more than just "wiring it up".

2.1 THE DEMONSTRATION CIRCUIT

As a demonstration of the feasibility of this approach to path finding, I built a small array of processors out of standard TTL parts. Figure 5 shows the circuit I used. As can be seen from the figure, there is not much to the "processor". It consists of one and two halves standard TTL packages, a few resistors, and an LED display. The circuit uses a 74161 as a four bit latch. The 74161 features the TC output, which is the logical AND of the four latch outputs and the E_T input. The four bits in the 74161 are the four arrow bits. The fifth bit is formed by one NAND and one NOR gate to indicate the blocked condition. Global control signals are circled. The two signals START and BLOCK are independent for each cell and are activated by momentarily grounding that node with a probe tip. Communication with neighbor processors enters this processor at the preset inputs of the 74161 and exits to the neighbors from the NAND gate at the right.

In operation, the circuit is quite simple. Initially, the CLEAR signal is taken low to clear all the block flip-flops. Then the maze walls are defined by selectively blocking some processors by grounding their BLOCK inputs. The LED decimal point lights in the blocked cells. Next, RESET is taken high for at least one clock cycle. This forces all communication wires between neighbor processors high and parallel loads all ones into the 74161, turning off the LED segments. This is a stable configuration, and will not change as the clock ticks. All communication wires stay high, and the latches keep parallel loading all ones because the TC outputs are high. Now suppose that somehow the processor to the right of this one changes out of its all ones state. Then its TC output goes low, causing the latch to stop parallel loading and causing the communication wires leaving the processor to go low. One of those wires enters this processor on the P_D input. At the next clock cycle, that low state is loaded into the D bit of the latch, turns on the right LED segment indicating a right pointing arrow, and prevents further parallel loading of the latch by forcing the TC output low. The result, then, is an indication on the LED for this processor that something happened to the right of it. Incidently, when the TC output of this processor went low, so did the outgoing communication wires, so on the next clock cycle, the other neighbors of this processor will be activated just as described above. Now, how did all that get

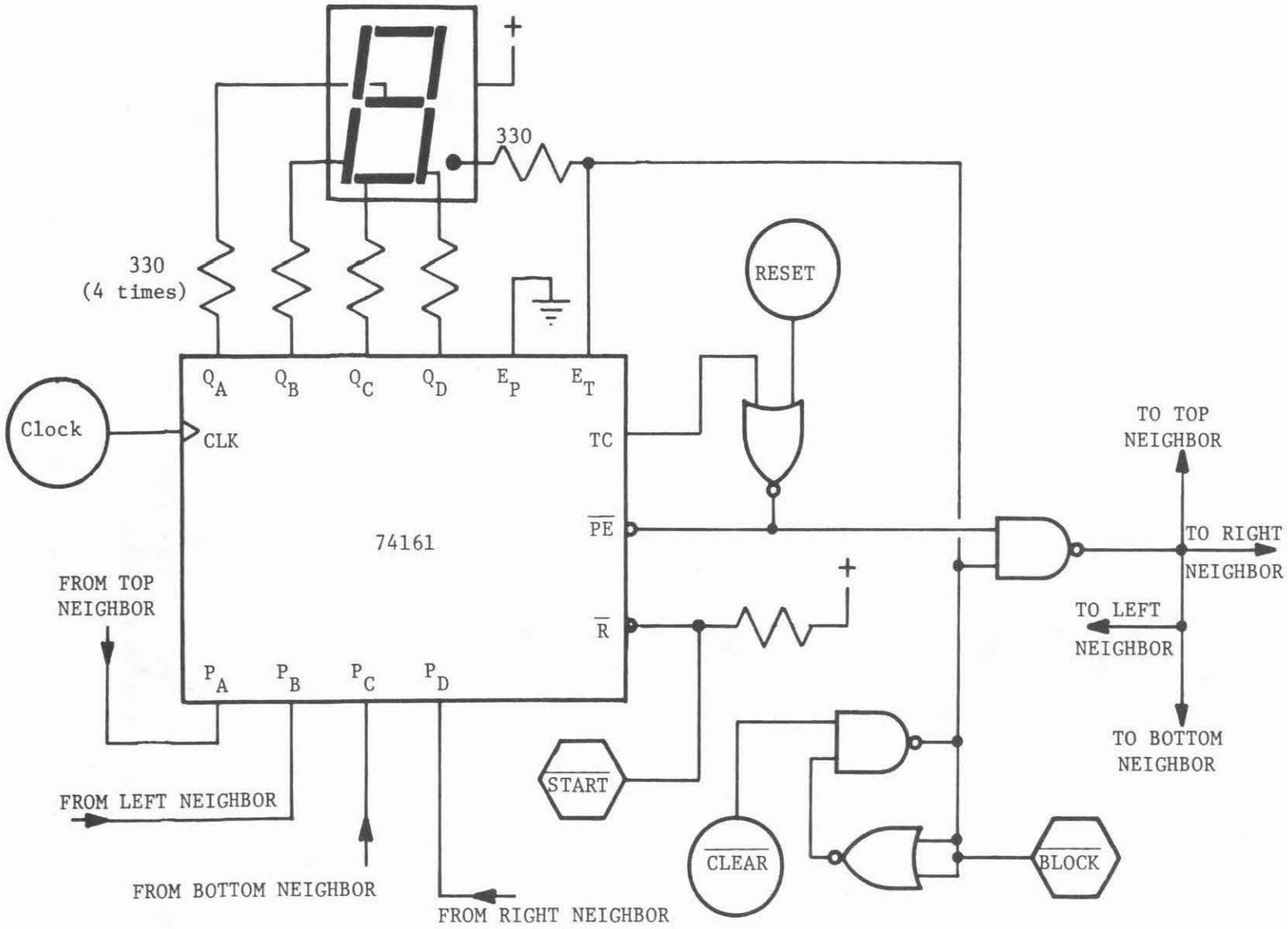

Figure 5. Schematic for Demonstration Circuit

started? Well, the START input on one cell was momentarily grounded, causing the latch outputs to go to all zero, turning on all four LED segments indicating that propagation started there, and causing that cell's communication outputs to go low. It is actually a very simple process that each processor in the array must execute. There is no computation in the numerical sense involved. Each cell simply passes on the propagating signal when it arrives, and records from which direction(s) it came.

When the propagation reaches the edges of the array, or can go no farther because of blocked cells, the action stops. What is recorded by the LEDs is actually the direction to go from each cell in the array to get back to the cell where it all started. The hardware has found the shortest path from the starting point to any other point in the array.

I designed this circuit purely for demonstration purposes. As such, tracing the path back is a visual process done by looking at the LED displays. If automatic trace back were desired, the five bits in each processor would be accessed as five bit words by a general computer which would then consider the processor array as a block of smart memory. It is an easy task for an ordinary computer to decipher the bits from each processor to find the path desired.

Before proceeding, consider what has happened to the computational effort required to reach this result. Time complexity of the algorithm has been dramatically improved. Now, rather than having a ***single processor*** advance the wavefront by stepping around the starting cell one cell at a time in an expanding spiral, the propagation takes place by activation of successive ***rings of processors*** surrounding the starting cell. At any given time, a number of processors directly proportional to the length of the path are actively working, rather than just one processor. The time required for the wavefront to expand out to the goal point is now directly proportional to the length of the path, not to the square of the length. Thus, the time complexity of the algorithm is now ***linear***, not quadratic, with respect to the path length. This result is expected -- a linear number of active processors can do in linear time what one active processor can do in quadratic time.

The circuit described above is so simple that it seems natural to lay out several copies of it on a silicon chip. That is just what was done for the design of the ***MAZER*** chip.

2.2 THE MAZER CHIP

The first step in designing the ***MAZER*** chip was to develop a NMOS circuit which performed the function of the demonstration circuit above. The stumbling block was the clocking scheme. Edge triggered latches are not as easy to come by in MOS as they are in TTL. Usually a set of multi-phase clocks are used to latch signals. This seemed to unnecessarily complicate the circuit, and a way around the problem was sought. The answer turned out to be easy. Just don't use any clocks!

On careful scrutiny of the operation of the demonstration circuit, one sees that clocking is really unnecessary. The only operation performed by each processor consists of waiting for the propagating wavefront to reach it, recording the direction(s) from which it came, and passing it along to its neighbors. One could imagine the array of processors as an array of mousetraps, each cocked and ready to fire. Each mousetrap is designed to fire as soon as any of its neighbors fire. Each mousetrap will store the direction from which its firing signal comes. At the end of the outward propagation process, which might always be allowed to propagate to the extremities of the array, the contents of each mousetrap cell's storage would then be the required arrows pointing in the direction of the shortest path from that cell to the start of the wave propagation process. This is a good visualization of the way in which the ***MAZER*** works.

Figure 6 is a conceptual logic design of a simplified ***MAZER*** cell. Not shown are all the mechanisms for accessing the cell, blocking the cell so that it becomes part of a "wall" in the maze, causing that cell to be the starting point of wave propagation, etc., but the mousetrap characteristic is illustrated. After the reset line has gone high to make all the flip flop Q outputs low, all signals which cross the cell boundary are low, and the system is stable in this state. Now, if for some reason one of the incoming signals goes high, the corresponding flip flop will be set. This causes the inputs to be disabled via the AND gates, and also causes the cell to generate a high going signal to each of its neighbors, triggering them in the same way. The flip flops remember from which direction the activation signal entered the cell, and reading them out by an accessing mechanism not shown gives the direction the maze solution takes as it passes through this cell.

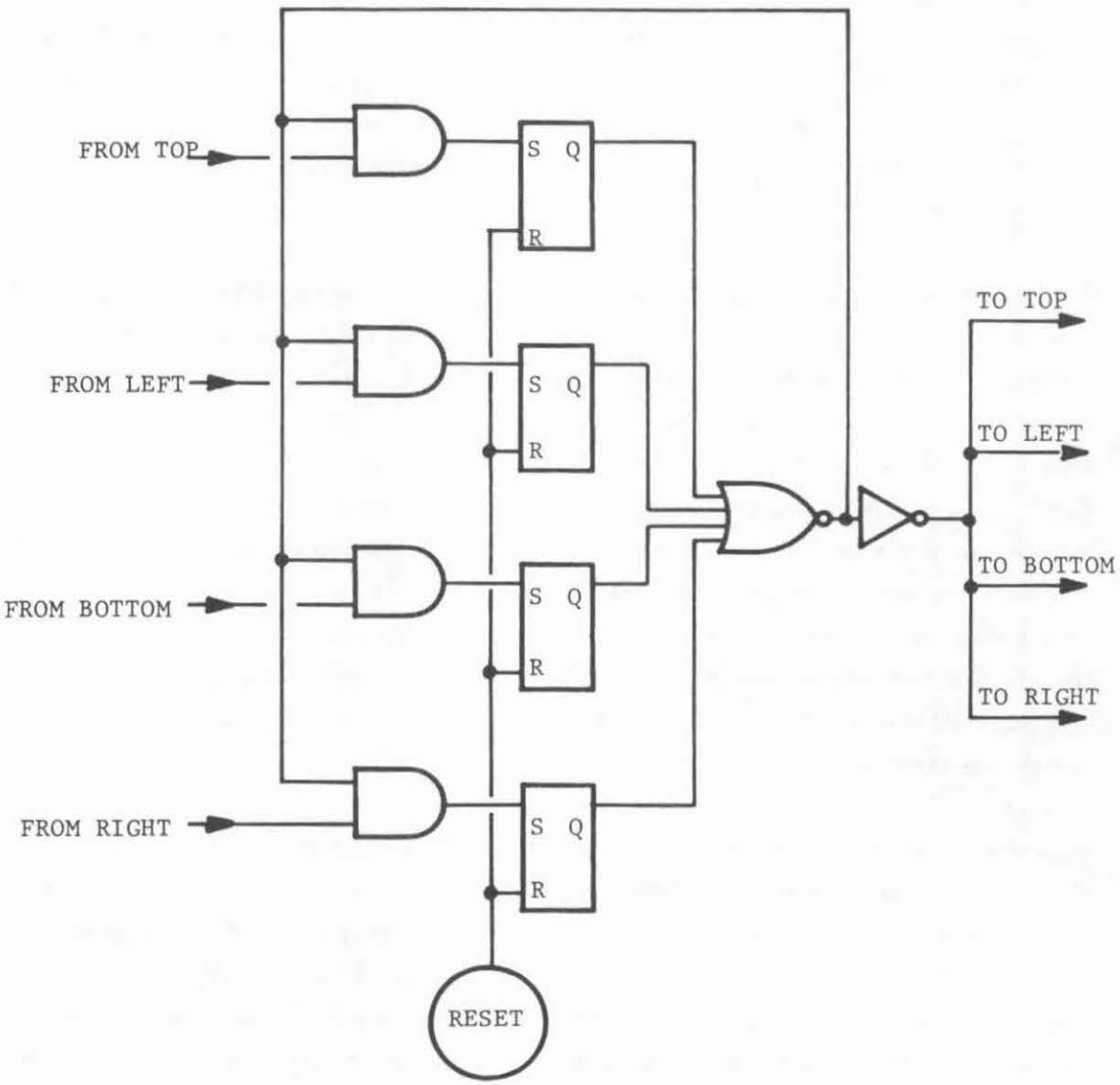

Figure 6. The Mousetrap Concept

Figure 7 is an actual schematic of a ***MAZER*** cell. Three of the AND gate/flip flop combinations of Figure 6 are seen here as transistor groups Q1-Q6, Q7-Q12, and Q13-Q18. The fourth direction is identified by the state where the cell has been triggered, and the other three flip flops are not set. The NOR gate and inverter are formed by Q19-Q25. Four bits of information are provided, as open drain outputs wire OR-ed with other cells on the chip. These bits are the three flip flop outputs plus a signal which indicates if the cell mousetrap has been "sprung". Transistors Q33-Q36 form a flip flop to store the blocked condition. ROW and COLUMN are addressing signals to select the cell for data readout, BLOCKing the cell to make it part of a maze wall, or STARTing the propagation process with this cell. RESET re-cocks the mousetraps, but does not destroy the blocked condition in the maze wall cells. CLEAR unblocks all the cells in preparation for a new maze. The other signals are communication paths to adjacent cells.

The complete ***MAZER*** chip contains sixteen processors arranged in a four by four array. Larger arrays can be assembled by arranging ***MAZER*** chips themselves in an array. Four wires come off each edge of the chip for the purpose of communication to adjacent chips. There are 15 additional wires which come off chip for data and control. Four are for data outputs, four are for address inputs, two are for power, and five are for the control signals BLOCK, RESET, CLEAR, START, and CHIP-ENABLE. A plot of the ***MAZER*** chip is shown in Figure 8. The four by four array of processor cells can be seen, surrounded by the 31 pads. Designed with Caltech's conservative design rules, the chip measures 2241 microns square, or about 8100 square mils.

2.3 MAZER CHIP TEST AND CHARACTERIZATION

The ***MAZER*** chip recently returned from fabrication and some preliminary results of testing are in. I have three chips which apparently contain no processing faults. The chips seemed to perform strangely, until I diagnosed the symptoms and found a basic bug in the ***MAZER*** processor circuitry. With the bug uncovered, I was able to circumvent the problem and continue to test the parts.

In order to understand the bug in the ***MAZER*** circuitry, examine Figure 7. The four data outputs of each of the sixteen processors, which come from the drains of transistors Q27 through Q30 in the figure, are bussed together onto four global data wires on the chip, DATA1 through DATA4. These wires run to the chip output pad circuitry, where there is a single pull up transistor on each of them. The important

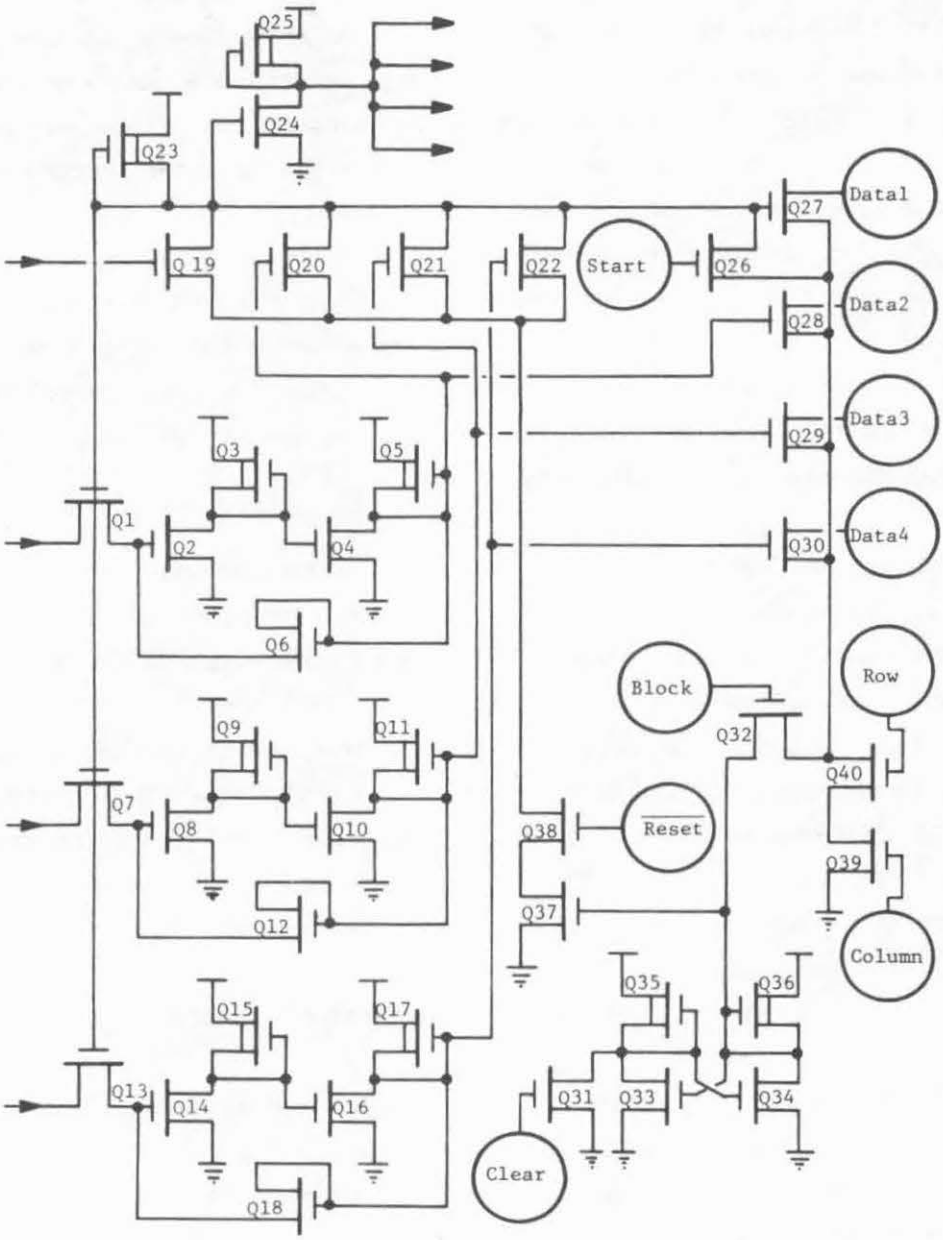

Figure 7. Schematic of a MAZER cell

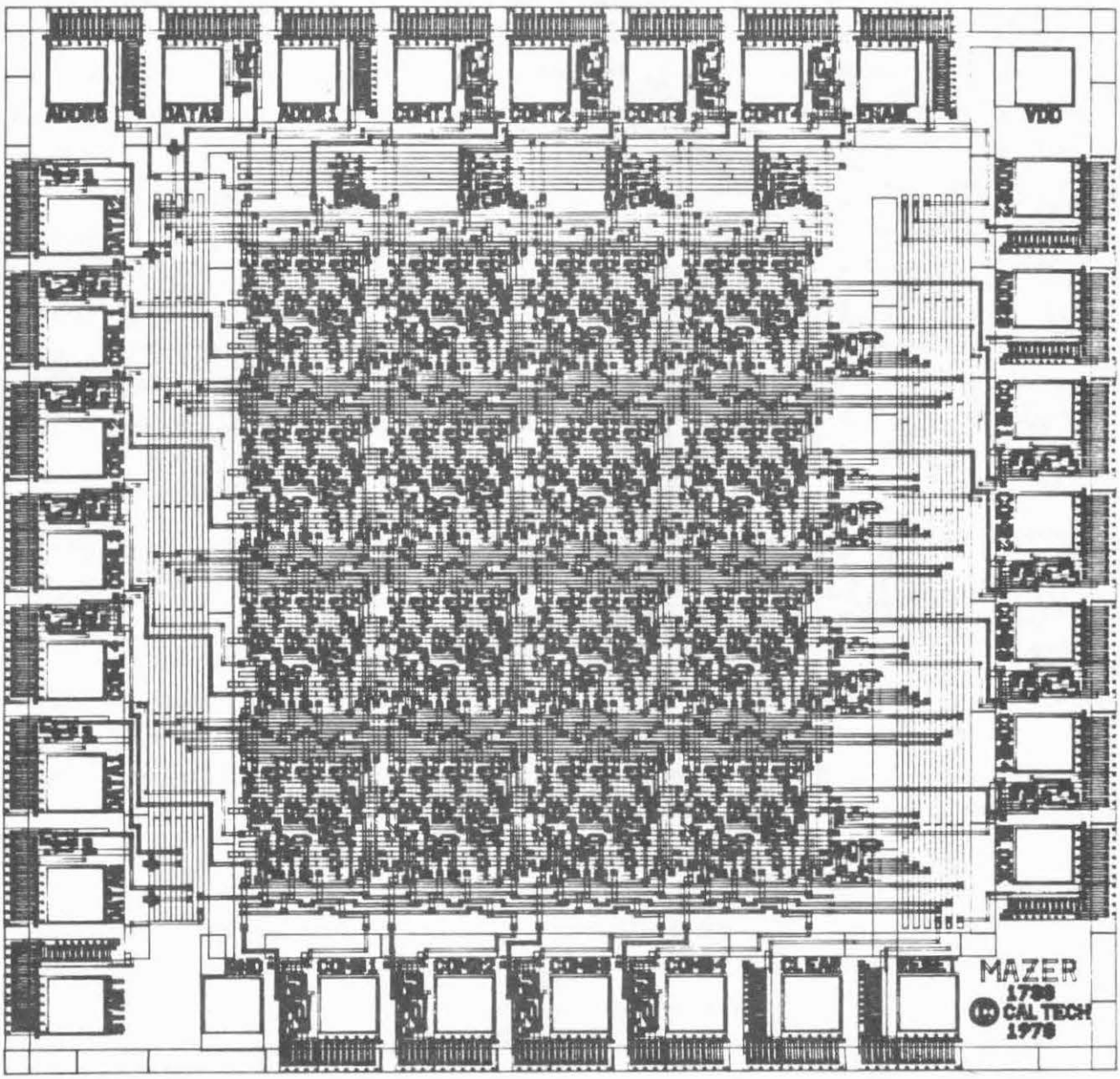

Figure 8. Plot of the MAZER chip

point to examine is the node in each processor which is common to the sources of the four transistors, Q27 through Q30. This node, which I will call the "enable node", is pulled low by the decoding transistors, Q39 and Q40, which are controlled by the addressing signals ROW and COLUMN. Since Q39 and Q40 are both turned on in only one of the sixteen processors, there should be a path from the enable node to ground only in that addressed processor. This prevents data in the non-addressed processors from pulling down on the data lines. However, things are not so simple. Suppose one of the data output transistors, say Q27, is on in the addressed processor, thus pulling down the DATA1 wire. Now, suppose that in another processor, which is not being addressed, both Q27 and Q28 are on. Since DATA1 is being held low by data in the addressed processor, Q27 in the second processor provides a path to ground for the enable node in that second processor. As a result, Q28 in that second processor can erroneously pull down the DATA2 wire. Since the addressed processor should not pull down DATA2, the result is that incorrect data appears at the output of the chip.

Fortunately, it is possible to retrieve correct data despite that problem. In the above example, notice that DATA1 is pulled down first by the addressed processor in the normal way. The bad data does not start to pull down on DATA2 until DATA1 is down, and even then, the string of transistors doing the pulling on DATA2 is longer than normal. The result is that good data shows up on the chip output pads about fifty nano-seconds before it is ruined by bad data caused by the sneak paths. Latching the good data in an external latch at the right time retrieves the correct state of the internal bits.

The operations of STARTing propagation and BLOCKing the cell are also affected by the unwanted paths between enable nodes and ground. Luckily, because of the high pull-up to pull-down ratio in the Q24/Q25 inverter, resulting in a very low switching threshold, STARTing can be performed normally. Apparently the sneak currents are low enough to prevent the inverter from switching in the non-addressed processors. However, so far I have been unable to individually BLOCK cells. The effect of blocked cells can nevertheless be tested by using the random distribution of blocked cells present after power-on. The CLEAR signal, which clears all blocked cells, operates normally.

In spite of the difficulties mentioned above, the test results are encouraging. All the arrows point correctly back to the cell where propagation starts. Some local

asymmetries sometimes are present, indicating that propagation proceeds a little more quickly in some areas of the chip than in others, but this result is expected with the asynchronous scheme used, and is negligible anyway. Access time, from chip enable to data output, is around 100 nano-seconds, about normal for a chip of this small size. A more detailed characterization of the chip remains to be completed.

3. HARDWARE FOR FINDING TWO-LAYER PATHS

The fact that the ***MAZER*** is limited to single layer paths limits its usefulness. The most immediate application for path finding hardware is in the area of printed circuit board design. However, single sided circuit boards are not very exciting. The step to two sided boards dramatically improves wireability and board density. Adding even more layers to the board improves density still more, but the additional effort does not buy nearly as much as the move from one to two sides. Thus, there was great incentive to develop a two layer path finder, with the specific goal of producing a routing machine for two sided printed circuit boards. This is the background for the design of the other integrated circuit to be discussed, the ***PATHFINDER***.

At first glance, it would seem that one need only construct a circuit that forms the topology of two ***MAZER*** chips laid on top of one another, with an additional arrow bit in each cell to indicate travel from one layer to the other. This strategy would work, except that it lacks some properties which have been found very desirable in the two layer environment. In what follows, terminology of the printed circuit board world will be used, with the understanding that other applications, such as interconnect wiring on integrated circuits, would have analogous features and terminology.

The first feature that would be missing in such a two-layer ***MAZER*** is the ability to block travel from one side of the board to the other independently from blocking travel through those cells without changing sides. Often it is desirable to prevent these holes in the board, or vias, from occurring in certain areas of the circuit board. Perhaps vias are to allowed only on a tenth inch grid, for example. Furthermore, vias sometimes can affect more than just the cell in which they occur. A via in one cell may prohibit the placing of a via in an adjacent cell. For all these reasons, an additional bit is required for via blocking in any proposed two layer path finding system to make it useful.

The second missing feature is much more disturbing. Designers of two layer circuit boards have long realized that it was advantageous to employ a tendency for wire runs that were mostly vertical to end up on one side of the board, and runs that were mostly horizontal to end up on the other side. This helps to avoid unnecessarily blocking channels for future wires. The tendency of a wire to choose one side of the board or the other depending on its orientation would be completely lacking in a straightforward two-layer ***MAZER***. Incorporating this preference into the basic path finding algorithm was an interesting problem, and the methods developed to solve it in both the traditional software implementations and the current hardware implementation will now be examined.

A way to achieve the wire location preference is to use a system of ***costs*** associated with travel from cell to cell through the array. One group to incorporate these costs into a standard software wire routing system was a group at Burroughs Corporation (5). Each cell stored one integer, but rather than storing ascending ordinal numbers on successive wavefronts in the propagation phase, as in the original algorithm, each cell stored the accumulated cost for reaching that cell from the starting cell. Suppose C(a,b) is the cost for expanding the wavefront from cell a to its neighbor, cell b. Then as the wavefront passes from cell a to cell b, the number stored in cell b is the number stored in cell a plus C(a,b). Since different costs might be encountered along different routes from the starting point to a given goal point, a number which has been previously stored in a cell might be overwritten if the wavefront reaches that cell from another direction with a lower cost than that achieved by the first contact with the cell. This is shown in Figure 9. Notice that the wavefront expands in exactly the same way as it did in the original algorithm, but now cells on the frontier are not necessarily all equally "distant" in terms of costs, as they were in the schemes described earlier. In the retrace phase of the algorithm, the numbers stored in the cells are used in a way similar to that described earlier. However, rather than searching neighbor cells for the next member in a reversed sequence, each step of the retrace involves searching for a neighbor of the current cell with a stored cost less than that of the current cell by the amount of the cost of propagating from that neighbor to this cell during the propagation phase. This does not necessarily result in the shortest path from point A to point B. What comes out instead is the least costly path between those two points, based on the cost function C(a,b). This is the meaning of the phrase "least costly" in the introductory paragraph of this paper.

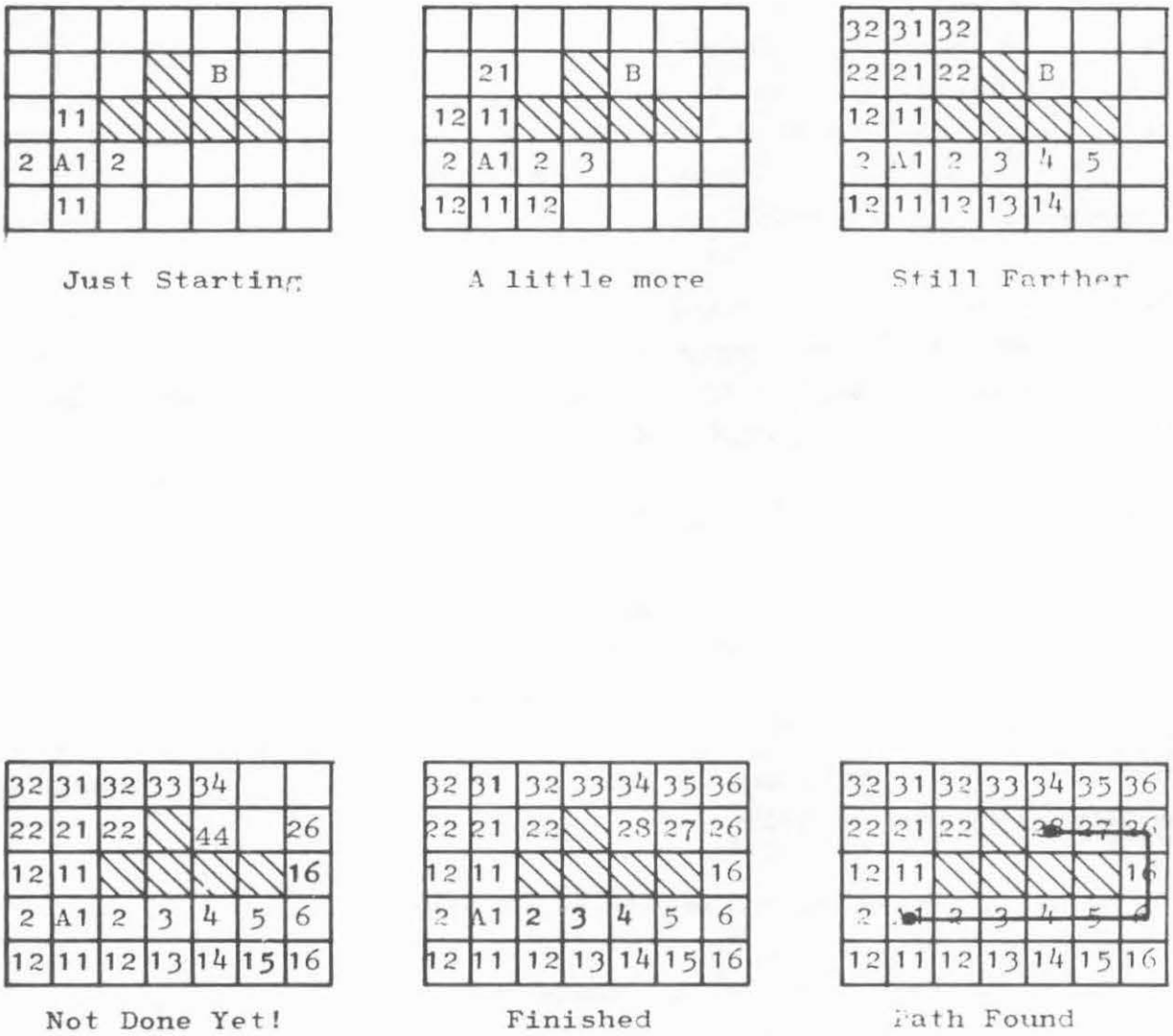

Figure 9. Path found by using a cost of 1 for travel in the horizontal direction and 10 for travel in the vertical direction.

The simplest cost function normally used consists of only three distinct costs. One cost is used for travelling in the "easy" directions, north-south on one side of the board and east-west on the other side, a second, slightly higher, cost is used for travelling in the "hard" directions, east-west on the first side and north-south on the second side, and a third, even higher, cost is used for travel "through the board", from one side to the other. Fancier schemes are possible. These involve reduced costs for travel near to and parallel to the edges of the board to increase utilization of that area, increased costs near component pins to prevent blocking future access to those pins, etc. The task of developing a cost function tailor-made to a particular circuit board can become quite an art.

Note, however, that using this cost function as a solution to the two layer situation brought back the same problems the original algorithm had, namely an unbounded number of bits of storage per cell, and global distribution of a numerical cost function. If there was to be any hope of building a chip comparable to the ***MAZER*** for two layer circuit boards, these problems had to be eliminated. To accomplish this, the ***MAZER*** was re-examined.

The scheme of using arrows instead of numbers seemed to be the way to go to limit the number of bits per cell. Using this method, however, required that during the propagation phase the expanding frontier of the wavefront must include only cells that were equally distant in terms of cost from the starting point, unlike the above two layer approach. This seemed to be at odds with the uniform, diamond shaped wavefront propagation described above.

The solution to the costs problem was to control the speed of wavefront propagation from cell to cell, rather than let it go at gate delay speeds. Consider the simple three cost system described earlier. If propagation could be allowed to proceed quickly in the "easy" direction, more slowly in the "hard" direction, and even more slowly in the "through the board" direction, the wavefront would meet the requirement that all cells on the frontier would be at an equal "distance" in terms of cost from the starting cell. Imagine a system where north-south propagation is easy on the top of the board and hard on the bottom. On such a board, a wavefront propagating from point A to a point B directly north of point A will reach point B on top of the board first, and will thus store arrows indicating a path which travels on the top side of the board back to point A. Similarly, if point B were to the east of point A, the wavefront, propagating more quickly in the east direction on the bottom of the board than on the top, would

cause point B to store arrows indicating retrace along the bottom of the board back to point A. No matter where point B was, the arrival of the wavefront would store information describing the least costly path back to point A.

During the time this solution evolved, some redundancy in the storage used in the ***MAZER*** made itself known. If the propagation phase left an arrow in cell A pointing to a neighbor cell B, indicating that retrace should proceed in that direction, that implied that there would be no arrow in cell B which pointed to cell A. Since that particular combination, adjacent cells pointing at each other, would never occur, there must have been some redundant information stored there, implying that a reduction in storage was possible. This was accomplished by moving the location of the arrow bits from inside each cell to between cells. Since each arrow then served the two cells between which it lay, the total storage required for the arrows was halved.

With these new modifications to the basic Lee-Moore algorithm, it was time to start designing the two layer chip.

3.1 THE PATHFINDER CHIP

The difficult obstacle in the design of the ***PATHFINDER*** chip was the method to use in controlling propagation speeds. What was required was a way to vary the speed over at least a ten to one range in each of three directions, the "easy" direction, the "hard" direction, and the "through the board" direction. Also, the circuitry could not be overly complex, nor could it involve many wires to the global environment. However, the required speed settings were related to the cost function described above. The cost function was something that was set by little more than educated guessing and experimentation. There was nothing very critical about the exact values of the costs. Only approximate settings were required. All of these considerations led the design away from a digitally controlled speed system, and towards an analog system.

The method employed relies heavily on the dynamic charge storage abilities of MOS circuitry. Figure 10 shows the set up for a simplified, one layer cell with its surrounding arrows, not showing the blocking or accessing circuitry. Each cell contains a capacitor of about 5 pF, or so. Before the start of the propagation phase, the capacitors are all precharged by means of the precharge transistor. With all the capacitors charged, all the arrow flip flops have both outputs held low. To start

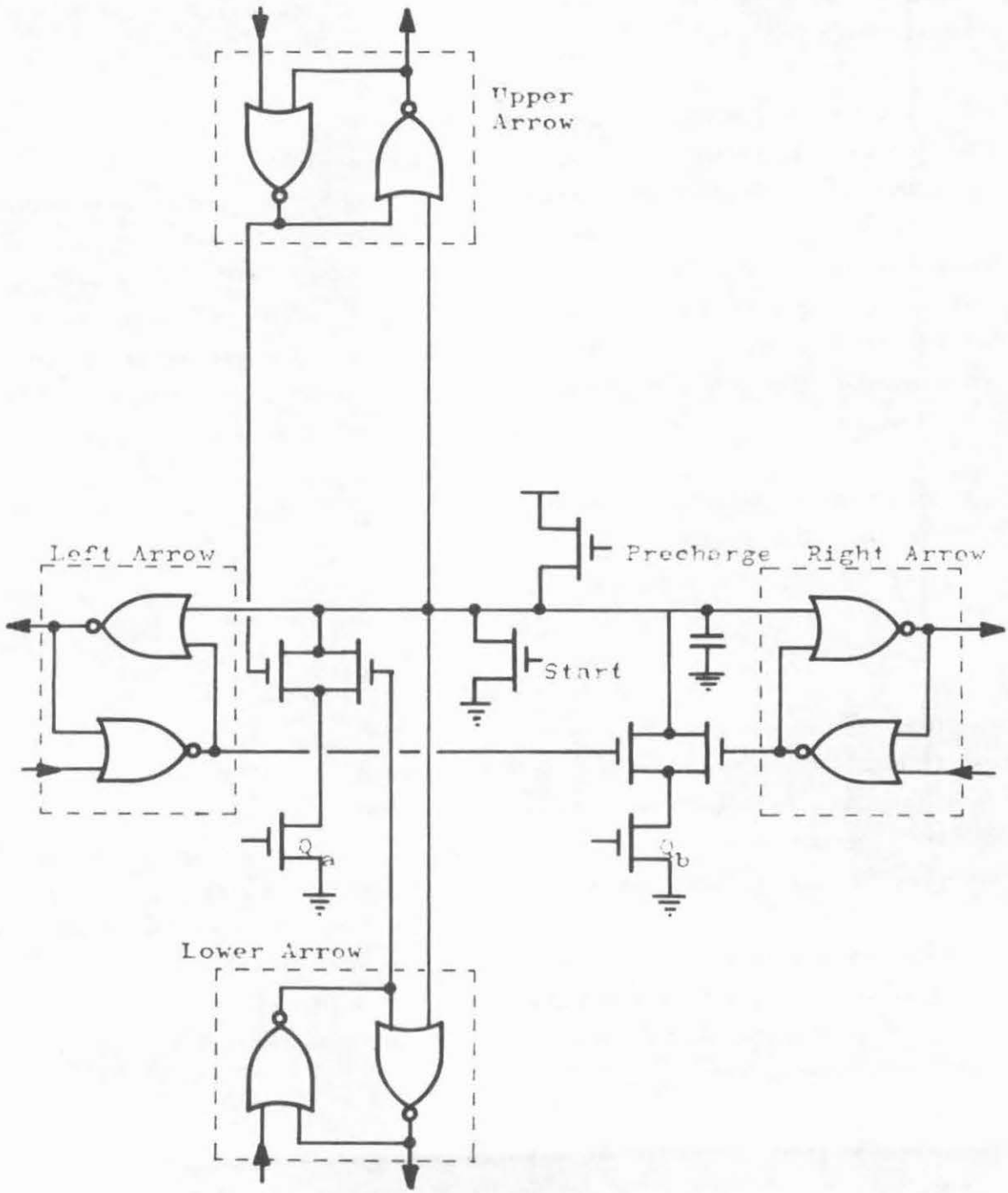

Figure 10. Simplified PATHFINDER Cell

propagation at a cell, that cell's capacitor is discharged. That action releases one side of the arrow flip flops surrounding that cell, causing those arrows to "point" to that cell with the discharged capacitor. The high outputs of the arrow flip flops then enter the neighbor cells, and begin discharging the capacitors there at rates determined by the voltages on the gates of Q_a and Q_b. When those capacitors are completely drained, the arrows surrounding those cells flip to point to the newly discharged capacitors, and the arrow outputs begin discharging capacitors in their neighbors. As the wavefront of activity propagates out, cells behind the frontier have completely discharged capacitors, cells ahead of the frontier have fully charged capacitors, and cells on the frontier have capacitors which are in the process of being discharged. The voltages on the gates of Q_a and Q_b are the crucial speed setting values. They are set by current mirror arrangements, as shown in Figure 11. There are three current mirror pads on the ***PATHFINDER*** chip, generating three control voltages for discharging the capacitors at the three rates required for the "easy", "hard", and "through" directions.

A feature included on the chip allows a small amount of local control over the cost function, to modulate the overall three costs described above. This consists of an additional pF or so of capacitance which can be switched on in parallel with the main capacitor in each cell. The time for propagating through a cell, and hence its propagation "costs", can be increased by connecting its extra capacitor before precharge and leaving it connected through propagation. The cost can be decreased by connecting the extra capacitor after precharge is over and disconnecting it again before propagation starts. These capacitor connections are switched on a cell by cell basis, controlled by a single bit in each cell. This makes it possible to increase costs near component pins, or to decrease costs near the board edges, etc., to reduce or increase the tendency for wires to end up in those areas. If circuitry had been included to discharge the extra capacitor when it was disconnected from the main one, additional levels of cost could be obtained by repeatedly connecting and disconnecting the extra capacitor between precharge and the start of propagation to remove more and more charge from the main capacitor, and thus reduce its discharge time. However, that extra feature was not included.

Figure 12 is a schematic of a two layer ***PATHFINDER*** processor, containing circuitry for the cells on both sides of the board as well as the arrow between them. The upper arrow and right hand arrow for each cell are arbitrarily assigned as belonging to that cell, while the lower and left hand arrows are considered to belong to the

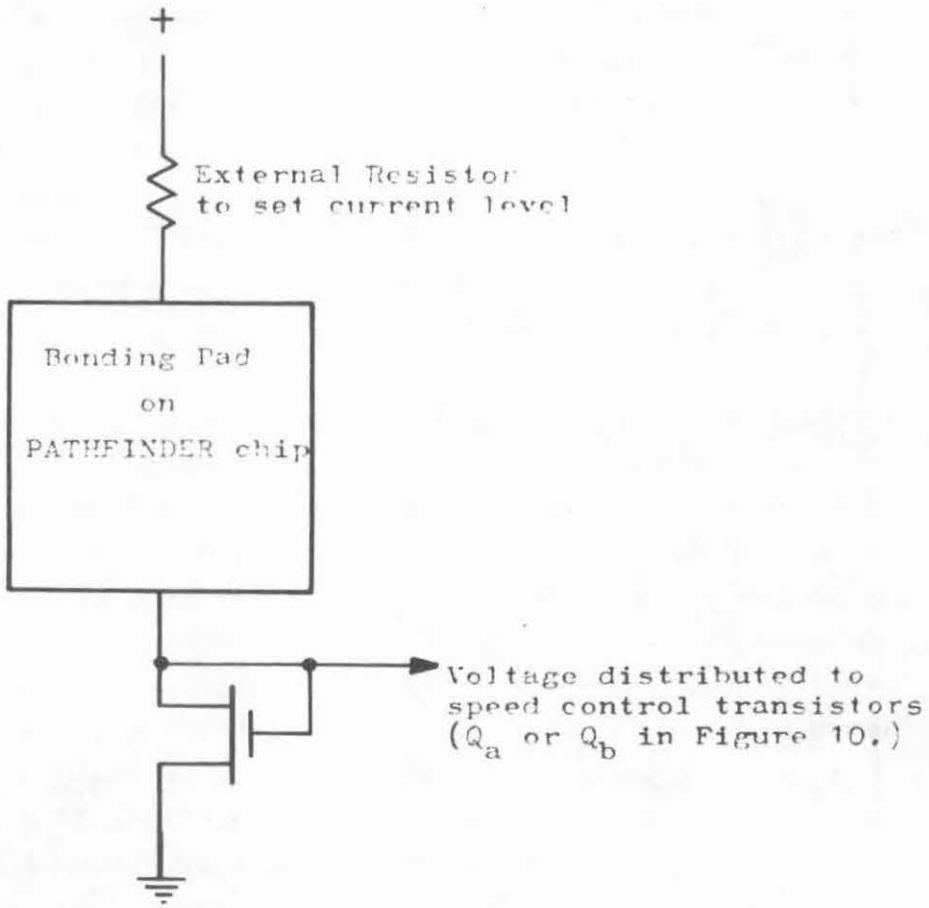

Figure 11. Example of a current mirror pad used on the PATHFINDER chip. There are three of these, one for each cost ("easy", "hard", and "through").

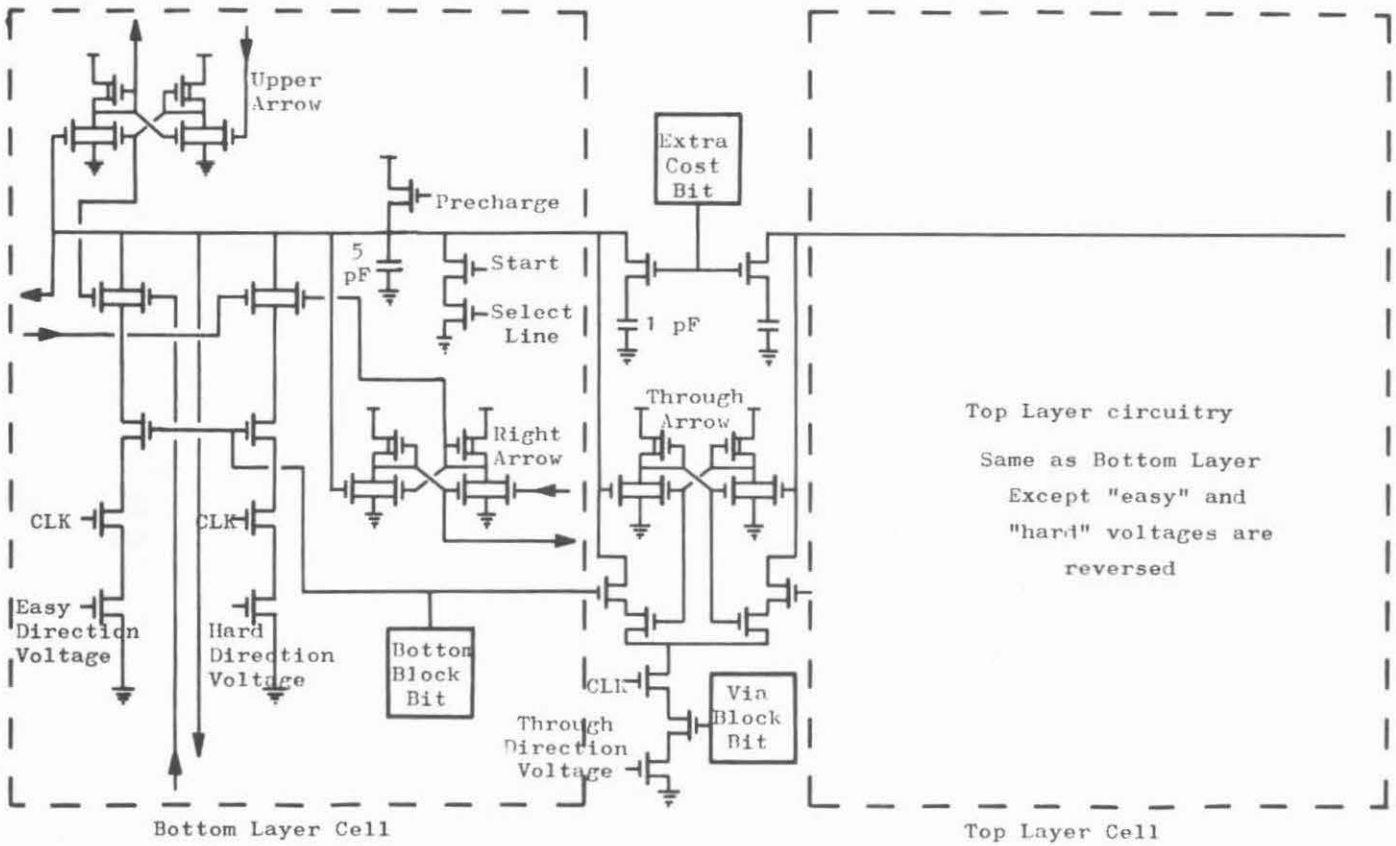

Figure 12. Schematic of PATHFINDER processor. Bit reading and writing mechanisms are not shown.

neighbor cells in those directions. The control storage bits are shown as boxes for simplicity. Actually the five arrow bits and the four control bits make up a nine bit word of what amounts to a standard static memory system, using the usual six transistor cell. Not shown are the two transistors which selectively link the flip flops to the word lines which run through all the bits, nor the select lines which control the gates of those transistors to do the addressing. Instead, the storage bits are shown located in reasonable places on the schematic to suggest their function in the circuit.

The circuit works just as described above for Figure 10, with the addition of the blocking controls and the switch to two layer operation. Having two layers merely means that three paths are present for discharging the capacitor, each controlled by a transistor whose gate voltage is set by one of the cost-setting current mirrors. The blocking control flip flops merely inhibit the appropriate discharge paths to prevent the discharge of the capacitor under the conditions which are to be blocked. The only remaining unexplained feature of the schematic is the signal labelled CLK. Remembering that this circuit was described as clock-less, one might wonder what the signal labelled CLK could be. In fact, it is a clock, but not in the usual sense.

The clock signal concerns a problem which has not been mentioned until now, having to do with chip boundaries in an array of cells composed of many chips. Because of the relatively large wiring capacitance associated with leaving one chip and entering the next, propagation from a cell on the periphery of one chip to its neighbor which happens to be on an adjacent chip will be much slower than propagation from cell to cell within the same chip. This can result in paths which have the problem shown in Figure 13, where the path takes every possible route to avoid crossing extra chip boundaries. The clock signal is an attempt to avoid this problem, by allowing the discharge process to occur only in short spurts. The clock is on for a few tens of nano-seconds, and then off for a hundred nano-seconds or so, giving the signals crossing from chip to chip time to settle. Letting the propagation process proceed only a little bit at a time like this slows down the system some, but should greatly alleviate the chip boundary problem.

Figure 14 shows a plot of the metal layer of the ***PATHFINDER*** chip. The chip contains a four by eight array of two layer processors. As with the ***MAZER***, the large processor arrays of several hundred processors on a side which are needed for useful printed circuit board work are built up by assembling ***PATHFINDER*** chips themselves in an array. Forty-eight of the seventy pads are devoted to chip to chip

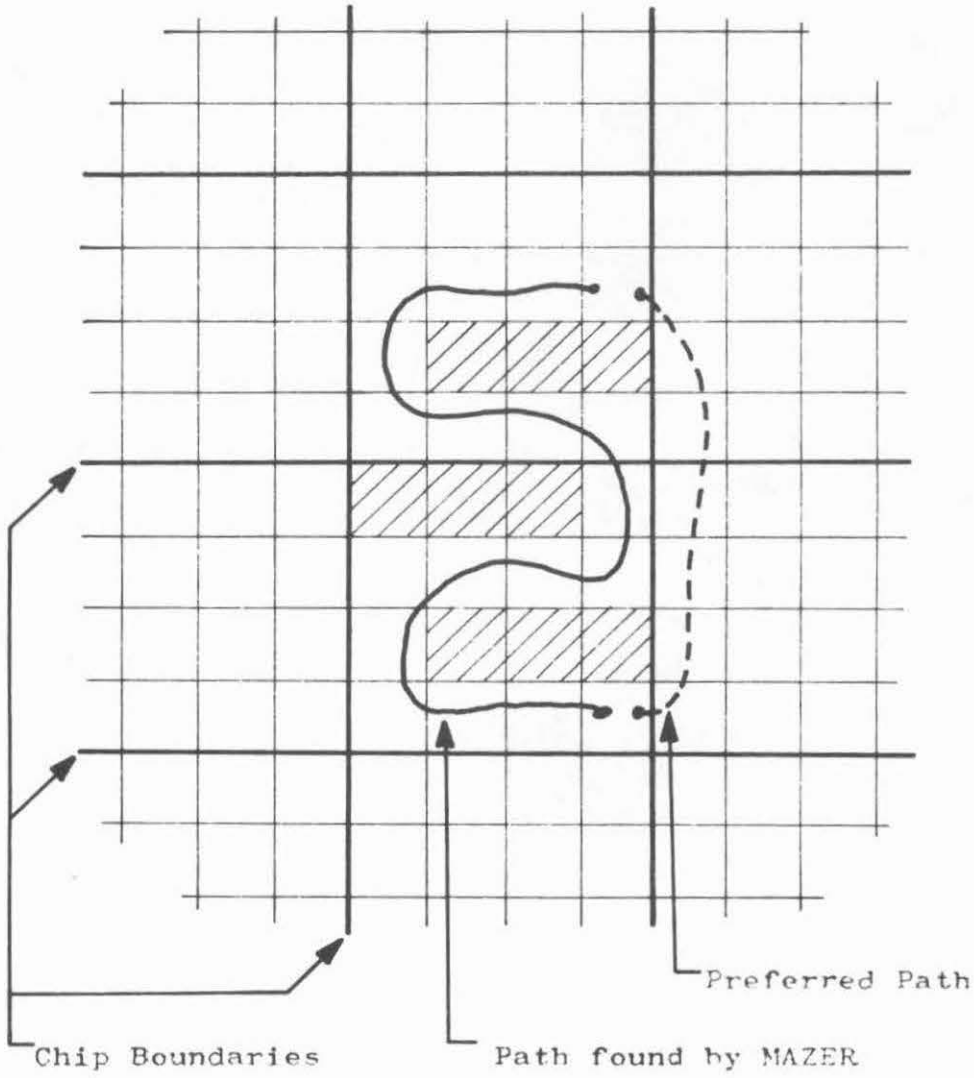

Figure 13. The problem posed by chip boundaries in large arrays

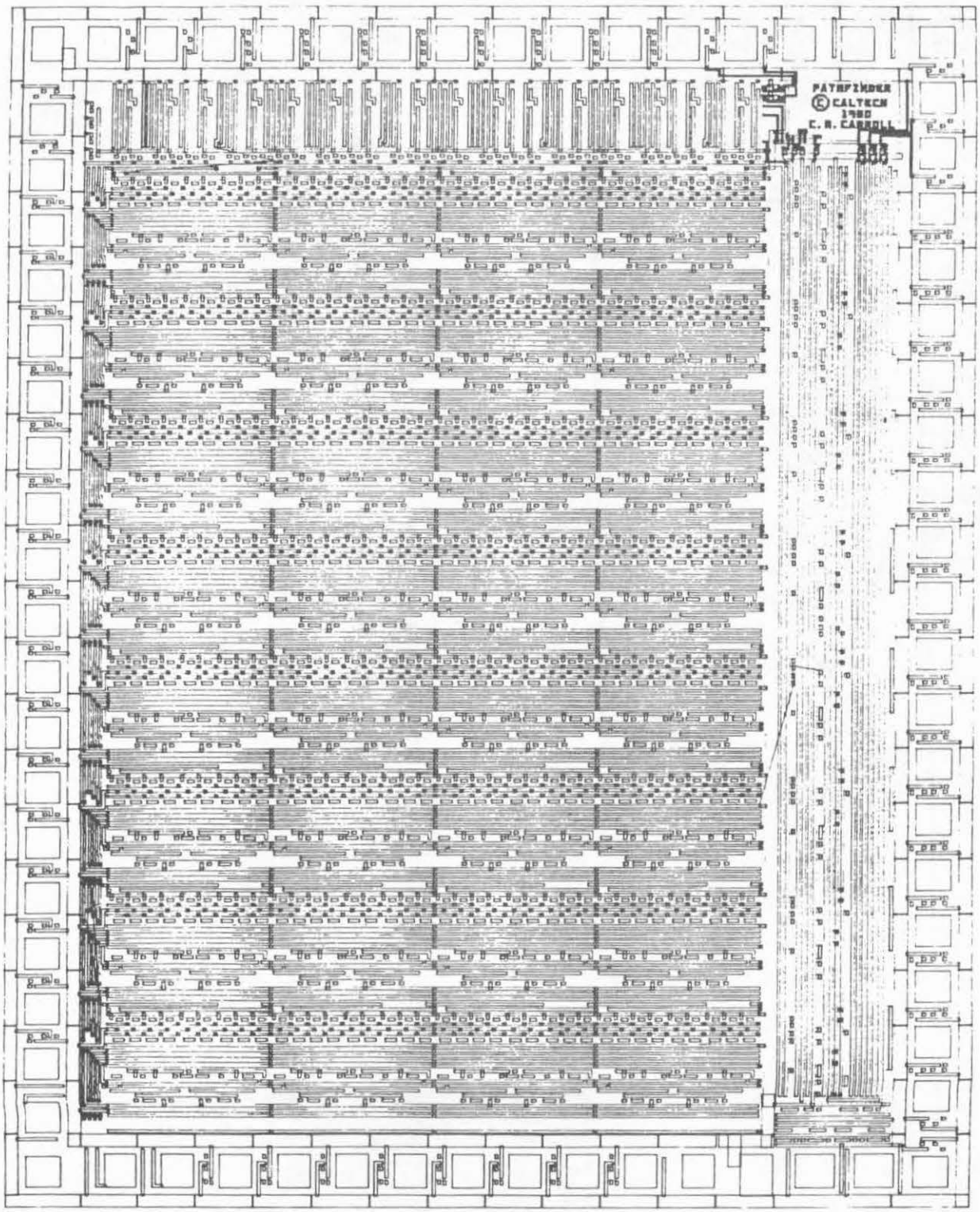

Figure 14. The PATHFINDER's metal layer

communication within the large array. The remaining pads consist of nine address pads, two power pads, two data I/O pads, and nine control pads, including the three current mirror cost-setting pads. Chip size is 3750 by 4875 microns.

3.2 PATHFINDER CHIP TEST AND CHARACTERIZATION

The ***PATHFINDER*** chip was included on the MPC380 run managed by Xerox PARC in the spring of 1980, and was also included on the M088 Mosis run managed by the Information Sciences Institute. From the two runs, I have received approximately twenty five copies of the chip. Only a few of the chips have been tested so far, however, because of difficulties in packaging the seventy pin circuit. Nevertheless, I do have one chip which is completely functional, and several others which work well enough to verify that the chip design is correct. Faults in the bad chips range from stuck bits to malfunctioning address decoders to complete failure, perhaps in the output buffers. Some of the chips have capacitors which fail to hold charge long enough to be useful. The capacitors in the one good chip, though, hold their charge long enough to keep the arrows "balanced" for about twenty seconds when carefully shielded from light. This is at least two orders of magnitude longer than required for successful path finding.

The chip has passed through several quantitative tests. Access time from chip enable to data out is around a microsecond, which is acceptable, though not noteworthy. An important item of interest is the operation of the cost-setting current mirrors. These perform very well. The range of control is excellent, with the normal external operating current between 100 and 1000 microamps. Characterizations of other quantitative aspects of the chip are not yet complete.

I have assembled a small microcomputer system to test the ***PATHFINDER*** chips and to operate them as a path finding system. With only one functional chip to use, no tests of the chip-to-chip communication strategy have yet been possible. However, I have written a true path finding program for the microprocessor which uses the one ***PATHFINDER*** chip to find paths through a four by eight grid. The program allows the user to set up any initial combination of blocked cells and blocked vias, start propagation from any cell in the array, and trace a path back to that starting cell from any of the other cells. The program displays the resulting path as well as the status of the control bits in the ***PATHFINDER*** chip on a terminal screen. In this implementation, three potentiometers, connected to the three current mirror pads on

the chip, are used to set the three global costs. The program can demonstrate the effect of changing costs on the path found between two points in the grid. For example, the user can cause the path to either skirt around barriers between the two endpoints, or to form vias and go over or under the barriers, depending on the settings of the three potentiometers. As more chips are tested and certified functional, this system can be expanded by connecting the good chips in an array to increase the size of the grid on which paths can be found.

SUMMARY

This paper has detailed the design of hardware which implements the computationally expensive parts of the Lee-Moore path finding algorithm. The progression of designs, leading to the ***PATHFINDER*** chip, show that this is a natural application of array processing. Applied to the problem of two sided printed circuit board wire routing, the use of the chips described here can reduce computation time from several hours to around a minute. However, this circuit is innovative not only because of its array processing aspect, but also because of its unusual use of analog variables. These cost-setting control voltages are not there to simply make the circuit work, as is the substrate bias on the chip, for example. Instead, the values of the control voltages are important parameters to the computation performed in the processor array. Changing the values of the voltages changes the result of the computation. This application of analog processing in a digital system may be just the beginning of a new design discipline combining the advantages of the analog and digital design worlds.

REFERENCES

1. E. Moore, "Shortest Path Through a Maze," Annals of the Computation Laboratory of Harvard University, Vol. 30. Cambridge, Mass.: Harvard University Press, 1959, pp. 285-292.

2. C. Lee, "An Algorithm for Path Connections and its Applications," IEEE Trans. Electronic Computers, Vol. EC-10, pp. 346-365, September, 1961.

3. S. Akers, "A Modification of Lee's Path Connection Algorithms," IEEE Trans. Electronic Computers, (Short Notes), Vol. EC-16, pp. 97-98, February, 1967.

4. Sutherland, Ivan, "A Better Mousetrap", Computer Science Department Display File #562, Caltech, March 8, 1977.

5. Slemaker, C., R. Mosteller, L. Leyking, A. Livitsanos, "A Programmable Printed-Wiring Router", Burroughs Corporation, Mission Viejo, California